The Report Committee for Ronald Craig Larcom
Certifies that this is the approved version of the following report:

Foveated Video Compression for Lossy Packet Networks

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Alan Bovik

Constantine Caramanis

Foveated Video Compression for Lossy Packet Networks

by

Ronald Craig Larcom, BSEE

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2010

Acknowledgements

I would like to thank Thayne Coffman, Alan Bovik, and Constantine Caramanis for their collaborations on this work. The research described in this Report was funded by the Army Research Laboratory.

December 5, 2010

Abstract

Foveated Video Compression for Lossy Packet Networks

Ronald Craig Larcom, MSE

The University of Texas at Austin, 2010

Supervisor: Alan Bovik

Unreliable networks can severely hamper transmission of video data. In applications requiring minimal latency, video frames must be compressed using intraframe techniques. We develop a video codec suitable for robot teleoperation over unreliable networks with high packet loss rates. The codec combines a foveated image compression algorithm, Embedded Foveation Image enCoding (EFIC), with a Forward Error Correction (FEC) code tuned to network performance. Foveation, or spatially variant image resolution, allows very high compression levels while preserving the most important image characteristics. By tightly integrating an FEC within the codec we are able to virtually eliminate dropped frames independent of the network protocol. We find that the new codec supports much higher video quality than another intraframe compression technique, Motion JPEG (M-JPEG).

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Video transmission enables a wide variety of activities, including entertainment, communication, security and robotics. In a raw data format, video data streams require an unfeasible amount of bandwidth. However, due to its inherent structure, video is highly compressible. Video transmission systems make use of coder/decoders (codecs), algorithms which compress video prior to transmission, possibly perform additional coding to detect and/or correct transmission errors, and restore transmitted video at the receiver.

A great number of codecs have been developed, addressing different video contents, compression requirements, transmission characteristics, and application needs. This report describes development of a new codec addressing a gap in current capabilities: video transmission supporting robot teleoperation over an unreliable packet based network. A codec to support this task must achieve significant levels of compression while guaranteeing low latency. Table 1 presents nominal operating characteristics for the expected use of the developed codec.

Table 1: Codec Operating Environment

Parameter	**Value**
Image Resolution	360x240
Raw Pixel Depth	24 bits (3 x 8 bit color)
Frame Rate	4-8 fps
Packet Maximum Transmission Unit (MTU)	1400 B
Transmission Rate	300 kbps – 500 kbps
Expected Packet Loss Rate	5-20 %
Required Compression	< 0.434 – 1.47 bits per pixel (bpp)

The most successful high compression codecs, such as most implementations of H.264/MPEG-4 Part 10, exploit motion compensation – transmitting occasional reference frames (I-frames) and motion updates used to produce intermediate frames (P-frames and B-frames). However, losing an I-frame can disrupt an entire Group of Pictures (GOP), resulting in extended periods of latency. Due to the expectation of low network reliability, the new codec will transmit individual frames using only intraframe compression. This approach minimizes the maximum latency caused by loss of a frame.

In order to receive the most operational benefit from available bandwidth we apply a foveated image compression algorithm to each video frame. 'Foveation' describes the property of varying spatial resolution – the selected compression algorithm provides fine detail about a selected fixation point, gradually increasing the coarseness of detail as distance from the fixation point increases. This approach matches the characteristics of the human visual system and is ideally suited for robot teleoperation due to providing high resolution at the operator's point of interest while still maintaining a low resolution periphery for context and situational awareness.

The packet-based network of interest is unreliable; the primary error mode for this network is packet loss – detected bit errors result in dropped packets. Additionally, the network protocol provides no mechanism of packet recovery. An adequate model of network performance is that any packet has an independent, identical likelihood of being lost entirely and a complementary likelihood of being transmitted error free. We are interested in developing a codec suited for operation when this likelihood of packet loss is high – between 0.05 and 0.2. The expected operating conditions indicate that multiple packets will be available for the transmission of each frame, between three and eleven. In order to make use of all available packets, the developed codec includes a mechanism

for allocating some bandwidth to a block error correcting code. This allows the codec, independent of the network protocol, to correct for dropped packets.

The new video codec has two distinguishing features – it relies on foveated image compression to provide extremely high intraframe image compression, and it integrates a block correcting error code to make transmission robust to packet loss. Due to these properties, we have dubbed the new codec 'RFCode,' short for 'Robust Foveated Encoding.' For the remainder of this report, the new codec will be referred to as the RFCode codec.

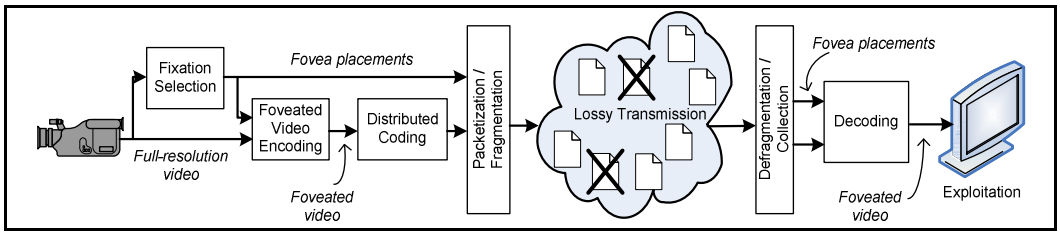

Figure 1: RFCode Codec System Diagram

Figure 1 presents a system diagram of the new codec. Foveated Video Encoding performs intraframe compression; the approach is described in detail in Chapter 2. A Fixation Selection algorithm, as described in Chapter 3, is used to configure foveation parameters. Distributed Coding, described in Chapter 4, applies a block error correcting code to compressed image data prior to packetization and transmission. In Chapter 5, we discuss our codec prototype and evaluation procedure. Chapter 6 presents results of our evaluation, and Chapter 7 concludes this report.

Chapter 2: Foveated Video Encoding

We selected the Embedded Foveation Image Coding (EFIC) algorithm, developed by Dr. Alan Bovik, to be the RFCode foveated encoding algorithm. EFIC employs an embedded wavelet coding approach, similar to Set Partitioning in Hierarchical Trees (SPIHT) [Said1996], in conjunction with a foveated weighting scheme. The EFIC algorithm is fully described in [Wang2001]. Below we present a brief overview of the approach, extracted nearly verbatim from [Wang2001].

Wavelet-based image coding algorithms have achieved great success in recent years. The success relies on the energy compaction feature of the discrete wavelet transforms (DWTs) and the efficient organization, quantization, and encoding of the wavelet coefficients. A class of embedded coding algorithms has recently received great attention. The most well-known algorithms are Shapiro's embedded zerotree wavelet (EZW) algorithm [Shapiro1993] and Said and Pearlman's set partitioning in hierarchical trees (SPIHT) algorithm [Said1996], which is an improved implementation of the EZW idea. Embedded wavelet image coding algorithms not only provide very good coding performance, but also have the property that the bitstream can be truncated at any point and still be decoded to yield a reasonably good quality image. This is a very attractive property that allows for scalable encoding and progressive transmission. Basically, the EZW and SPIHT encoders try to order the output bitstream, such that those bits with greater contribution to the mean-squared error (MSE) between the original and the compressed images are encoded and transmitted first. In other words, the progressive encoding scheme intends to minimize the MSE at any bit-rate. HVS features are not considered.

The goal of [Wang2001] was to design an embedded foveation image coding (EFIC) system, which tries to order the output bitstream, so that those bits with greater contribution to the foveated visual distortion are encoded and transmitted first. In other words, it is designed to optimize foveated visual quality at any bit-rate.

The wavelet coefficients at different subbands and locations supply information of variable perceptual importance to the HVS. In order to develop a good wavelet-based image coding algorithm that considers HVS features, we need to measure the visual importance of the wavelet coefficients. In [Watson1997], psychovisual experiments were conducted to measure the visual sensitivity in wavelet decompositions. Noise was added to the wavelet coefficients of a blank image with uniform mid-gray level. After the inverse wavelet transform, the noise threshold in the spatial domain was tested. Based on the heuristic fit developed for this data, a foveation-based sensitivity mask in the DWT domain can be determined.

The construction of the foveation-based sensitivity mask can be viewed as two stages in cascade. In the first stage, each wavelet subband is assigned a uniform base importance value according to wavelet decomposition level and orientation. In the second stage, nonuniform weights developed each point's equivalent distance from the foveation point in the spatial domain are applied to the subbands, resulting in a space-variant error sensitivity mask in the DWT domain. In Figure 2 we show the error sensitivity mask for a viewing distance equal to image width.

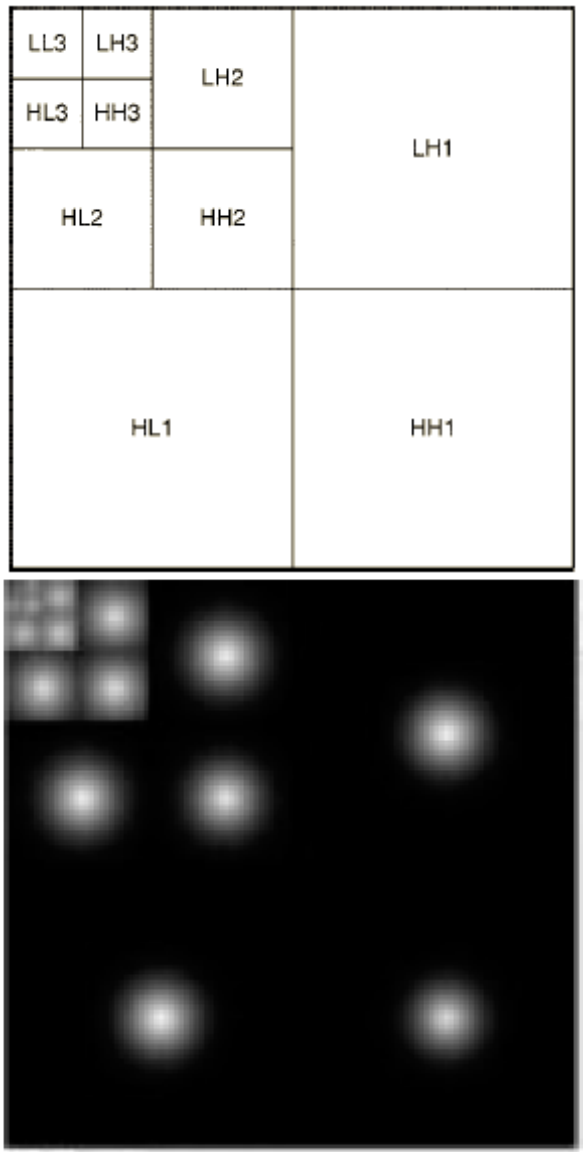

Figure 2: From [Wang2001]: DWT decomposition structure (left) and foveation-based error sensitivity in the DWT domain (right).

The DWT domain may be divided into subbands (such as LL3, HL1, LH1, and HH1) which describe the horizontal and vertical scale of features, as well as spatial locations at each scale (indicated by location within the appropriate square in the left image of Figure 2). For example, the LL3 subband contains the coarsest scale (low frequency) features of the image, while the HH1 subband contains the finest scale (high

frequency) features. The HL1 subband contains features with fine horizontal components and coarse vertical components, such as horizontal edges.

Within the DWT domain, the space variant mask (Figure 2 right image) indicates the amount each potential feature can contribute to perceptible error. We see that nearly all coarse features are important; all of the LL3 subband and most of the HL3, LH3, and HH3 subbands have high (white) values. As scale becomes finer, only those features closer and closer to the center of the image have any error sensitivity. In the finest scale, HH1, less than one ninth of the image makes any significant contribution to perceived error. The rest of the wavelet coefficients within this subband can be disregarded.

EFIC takes advantage of this property of perceptible error to compress images by:

1. Transforming the image into the wavelet domain. The image is convolved with high-pass and low-pass analysis filters and downsampled to decompose it into appropriate subbands.
2. Scaling the wavelet coefficients by the error sensitivity mask. This weights each coefficient relative to its contribution to error based on location and scale.
3. Encoding the scaled coefficients using a modified SPIHT encoder. This compression algorithm is described below.

The main objective in embedded wavelet image coding is to choose the most important wavelet coefficients to be encoded and transmitted first. The importance of a coefficient in EZW and SPIHT depends on its contribution to the MSE distortion. The coefficients with larger magnitudes are more important. The strategy is ordering the coefficients by magnitude and transmitting the most significant bits first. Assume that the wavelet coefficients have been ordered according to the minimum number of bits required for its magnitude binary representation. The schematic binary representation is shown in Figure 3. The most effective order for progressive transmission is to

sequentially send the bits in each row, as indicated by the arrows. In order for the decoder to understand the meaning of the bits, we also need to encode and transmit the coordinates of the wavelet coefficients along with the magnitude bits. It has been observed that the wavelet coefficients which are less significant have structural similarity across the wavelet subbands in the same spatial orientation. The zerotree structure in EZW and the spatial orientation tree structure in SPIHT capture this structural similarity very effectively.

In EZW or SPIHT encoder, the wavelet coefficients are scanned multiple times. Each time consists of a sorting pass and a refinement pass. The sorting pass selects the significant coefficients and encodes the spatial orientation tree structure. A coefficient is significant if its magnitude is larger than a threshold value, which decreases by a factor of 2 for each successive sorting pass. The refinement pass outputs one bit for each selected coefficient, as indicated by the arrows in Figure 3. An entropy coder can be used to further compress the output bitstream. SPIHT performs better than EZW in terms of reconstructed image quality. By applying a foveation-based sensitivity mask to the DWT domain data, EFIC modifies the SPHIT algorithm to order coefficients based on their contribution to perceived error, rather than just MSE.

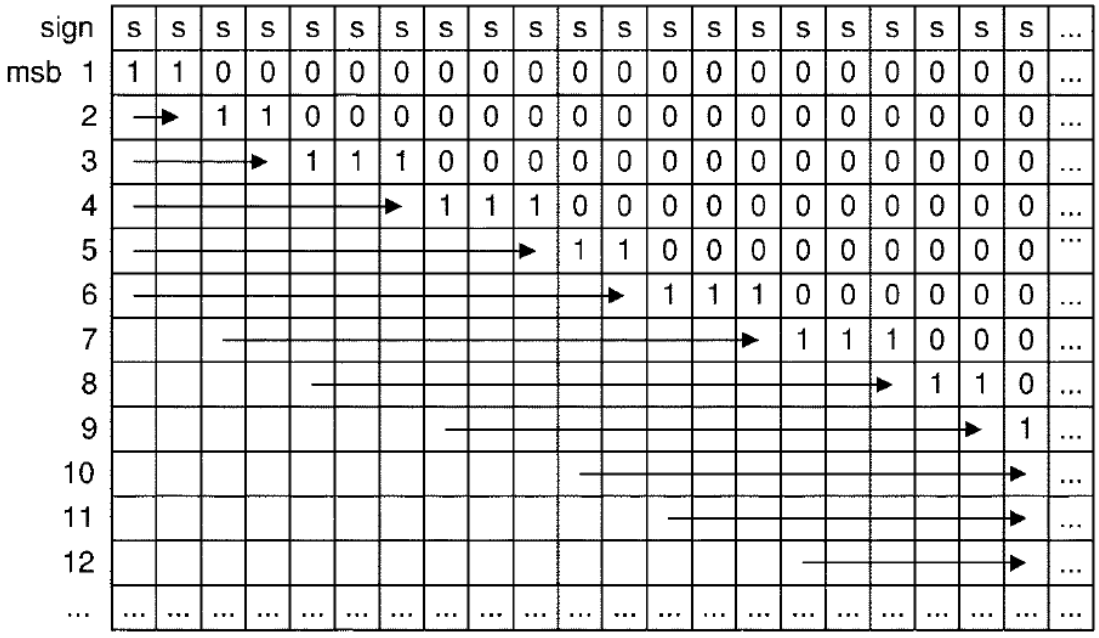

Figure 3: From [Wang2001]: Binary representation of magnitude-ordered weighted wavelet coefficients in modified SPIHT algorithm.

The output bitstream of the modified SPIHT encoder, together with the foveation parameters, is transmitted to the communication network. At the receiver side, the weighted wavelet coefficients are obtained by applying the modified SPIHT decoding. The importance weighting mask is then calculated in exactly the same way as at the sender side. Finally, the inverse weighting and inverse wavelet transform are applied to obtain the reconstructed image.

The above description covers the [Wang2001] development of the EFIC algorithm for black and white imagery. We extended the algorithm to support RFCode's requirement for color video transmission. To transmit a received RGB image we first transform the image into the YUV intensity-chrominance color space.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

We then apply the EFIC encoding algorithm to each channel independently, creating individual Y, U, and V, datastreams for transmission. Due to redundancy between the color channels, most visual information in an image is contained within the intensity channel (Y). Based on this observation, we use the common image compression strategy of allocating most bandwidth to the intensity channel. Due to the progressive encoding nature of the EFIC algorithm, we can truncate the color channels at any point by simply not transmitting additional wavelet coefficient bits. We allocate 80% of our available bandwidth to the Y channel, and 10% each to the U and V channels. This asymmetric bandwidth allocation had significant positive effects on EFIC video quality.

Chapter 3: Fixation Selection

The EFIC algorithm, described above, can support a variety of foveation parameters, such as fixation (area of high resolution) location, fovea size, and rate of resolution fall-off. One of the most important parameters operationally is the fixation location. By adjusting the wavelet weighting mask, the RFCode codec can place the fixation anywhere within the image frame.

In many applications, the logical solution is to maintain a constant fixation at the image center (this strategy was used for the majority of images contained within this report). One such application is transmission of video from a user-controlled pan-tilt-zoom sensor. In this case, the user places the fixation point over objects of interest by steering the sensor. However, in other applications it may be desirable to move the fixation throughout the video due to a fixed orientation camera. A mobile fixation can be controlled either by user feedback or by automatic algorithmic selection.

We have demonstrated the feasibility of automatically selecting fixation locations for the RFCode codec using an existing fixation finding algortihm. We integrated the RFCode prototype codec with the Gaze-Attentive Fixation Finding Engine (GAFFE) [Rajashekar2008], developed at the University of Texas, Austin. Within this new experimental system, GAFFE functions as a fovea placement algorithm. As shown in Figure 4, GAFFE processes the original full resolution image to select a fixation point. The selected point is used by the RFCode codec when foveating and encoding the source image.

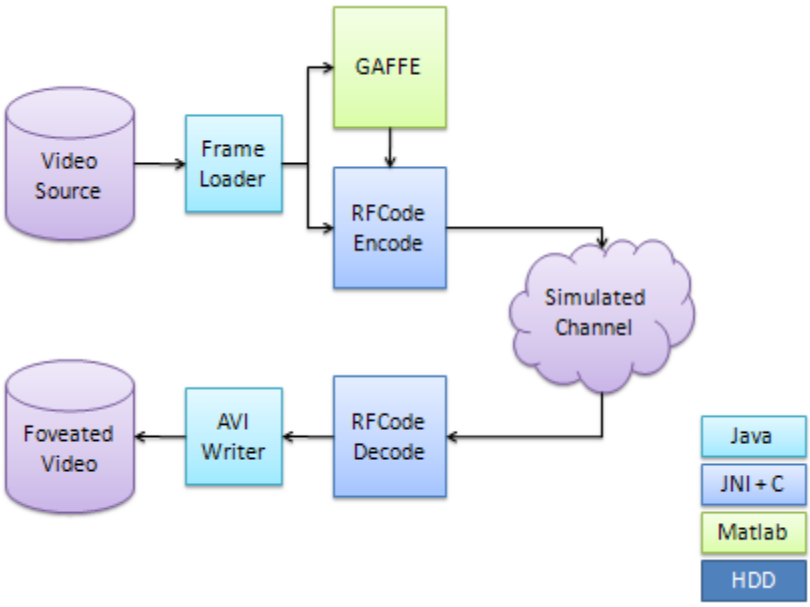

Figure 4: Automatic Fovea Placement Block Diagram

GAFFE selects fixation points by calculating and weighting four low-level image features: luminance, contrast, bandpass luminance, and bandpass contrast. Locations with high values of these features have been shown to be statistically more likely to attract human gaze during gaze tracking experiments. The GAFFE algorithm makes no assumptions about video content or the intended application.

Using the experimental setup shown above, we have recorded several test videos with automatic fovea placement. Figure 5 shows sample frames from three videos. In general, GAFFE finds interesting points near the subjective 'item of greatest interest.' Throughout the coastguard and container ship sequences the fixation point jumped between various locations on the ships' superstructures. The foreman sequence demonstrates the importance of context and application in selecting fixation points. To a human viewer, the areas of greatest interest are the speaker's eyes and lips. However,

GAFFE tends to predominately select interesting geometrical primitives either in the background or on the speaker's hardhat.

Figure 5: Sample images with automatic fovea placement (fixation marked in green), compressed to 0.25 bpp.

Depending on the intended use of the RFCode codec it can support either a general fixation selection approach such as GAFFE, or an application specific selection approach which cues off of higher level image details such as targets of interest. As

demonstrated, the underlying RFCode codec is capable of transmitting a video stream with a dynamic fixation location.

Chapter 4: Distributed Coding

We selected the Reed-Solomon (RS) error correcting code to provide resilience to packet loss. RS codes are popular for encoding bitstreams prone to burst error, or for packet based transmission due to their efficient computation and ability to recover whole word errors. RS codes correct errors based on multi-bit words; multiple bit errors within the same word are as easy to correct as a single bit error. This in contrast to a binary code, such as a Hamming code, which is better for correcting multiple single bit errors spread through multiple words. In the application domain of interest, we expect our errors to be tightly grouped – the most common error source is packet loss, in which case an entire word is in error (we call the missing data an 'erasure').

RS codes were first introduced in [Reed1960]; they are a family of non-binary cyclic codes with m-bit words ($m > 2$). RS codes recover entire words at a time, regardless of the number of bits corrupted within the word. This is why RS codes are preferred for bursty or packet-based channels. For a particular word-size, the code can support a total of n code symbols with k data symbols. We will use the notation RS(n,k) to described these codes. RS(n,k) is subject to the inequality:

$$k < n < 2^m + 2$$

Like binary codes, RS codes are based on polynomial multiplication and factorization. Unlike binary codes, an RS code utilizes arithmetic over the Galois Field GF(2^m), based on word size. From abstract algebra, a Galois Field contains a finite number of elements (2^m in our case), and is closed under addition and multiplication. A particular code is defined by a 'generating' polynomial of degree equal to the desired parity, (n-k). The polynomial's coefficients are members of GF(2^m). To encode, a message of k words is converted into a k degree polynomial, then multiplied by X^{n-k} (the

words are upshifted by the number of parity words). Parity is generated by calculating the remainder of the message divided by the generating polynomial. On the receiver end, errors are detected by identifying transmitted polynomials which do not share roots with the generator polynomial. An excellent primer providing more detail on the mathematics of RS codes is [Sklar2001].

In order to select an appropriate RS encoding for RFCode we needed to select a word size, message length, and maximum parity. Based on an analysis of the expected RFCode operating environment (see Table 1), one can see that RFCode must encode each frame in between three and eleven packets, parity included. This analysis is dependent on the assumption that transmitting packets at the Maximum Transmission Unit (MTU) is preferable for network performance.

At the expected bandwidths and packet characteristics, it is clear that RFCode will require RS codes with $k < n << 20$, without significantly affecting our coding flexibility. Therefore, we must select $m > 4$. Due to the convenience of working with bytes, we selected the word size $m = 8$. In particular, we chose to use an implementation of RS(255,223), a popular byte code that allows transmission of up to 32 parity bytes and a maximum of 255 packets. This code provides us the flexibility to encode using any total code size $n < 255$, and data size $k < 223$. In order to this, we simply zero-pad the initial message to form a 223 byte code word, and transmit only the desired n bytes (the non-padded code word and n-k parity bytes). In the frequent case that the desired n-k parity is less than 32, the receiver zero pads the data portion and the untransmitted parity portion, and identifies those parity bytes as errors.

In a packet based scheme, RS codes can be configured to augment an initial message of k packets with an additional (n-k) parity packets. Provided that the position of erasures in the message can be correctly identified, the entire message can be recovered if

no more than (*n*-*k*) packets are dropped. If each packet has an independent likelihood of being dropped, *d*, with $d < 1.0$, the number of packets dropped in a group of *n*, *D*, forms a binomial distribution. The probability of successfully decoding a transmitted message is:

$$P(D \le n-k) = \sum_{i=0}^{n-k} \binom{n}{i} d^i (1-d)^{n-i}$$

This approach gives us a very flexible error-correcting code capable of providing all expected levels of parity protection. There are a variety of strategies for selecting the best parity protection level. The first we explored is setting the parity level *p* (referred to as $n - k$ above) to maximize the expected number of data bytes received, for fixed *n* and known loss rate *d*. The expected data received is the packet size (*MTU*) times the number of data packets ($k=n-p$) times the probability of successful transmission given above, giving the optimal *p* as:

$$\hat{p} = \arg\max_p (MTU(n-p) \sum_{i=0}^{p} \binom{n}{i} d^i (1-d)^{n-i})$$

As an example, when six packets are available for a single frame (n=6), and at a loss rate of 0.2, the optimum parity is 2. In this case, the RFCode codec will employ a RS(6,4) code. The packet structure will look like Table 2. I_n indicates a packet's header information (packet number, parity information, etc.), B_n an image byte, and $P_{j,n}$ the *j*th parity byte corresponding to image bytes *n* .. *n*+3.

Table 2: Example RFCode Packet Structure

Packet 1		**Packet 2**		**Packet 3**		**Packet 4**		**Packet 5**		**Packet 6**
I_1		I_2		I_3		I_4		I_5		I_6
B_0		B_1		B_2		B_3		$P_{0,0}$		$P_{1,0}$
B_4		B_5		B_6		B_7		$P_{0,4}$		$P_{1,4}$
…		…		…		…		…		…
B_{3996}		B_{3997}		B_{3998}		B_{3999}		$P_{0,3996}$		$P_{1,3996}$

The above strategy is known as Equal Loss Protection (ELP). Table 2 describes an allocation in which each packet is either entirely data or entirely parity. In the illustrated allocation bytes B_0 to B_3 have the same amount of parity protection as bytes B_{3996} to B_{3999}. As discussed above, this parity allocation strategy is optimal to maximize the expected number of bytes recovered, and each data byte is assigned equal importance.

The EFIC algorithm, described in Chapter 2, uses a progressive compression algorithm – the encoded bit stream can be truncated at any point and a partial decoding performed. During this decoding process, each decoded bit has a greater impact on image distortion than any subsequent bit. From this perspective, it is intuitively clear that error correcting code (ECC) strength (alternatively number of parity bytes) should be biased towards the earlier message bytes. Such a biased allocation of ECC strength is known as Unequal Loss Protection (ULP).

For the RFCode codec we developed an ULP approach similar to that described in [Mohr1999], which also addressed transmitting a progressive encoding over a packet loss channel. Bandwidth and network MTU constrain RFCode video transmission to n packets of L bytes. In the ULP strategy, we divide the available packet data into L streams of n bytes each by grouping the l^{th} byte of every packet into the l^{th} stream. Each

stream contains 1 to n bytes of message data (taken sequentially from the EFIC compressed image) and f_l bytes of parity coding. We seek to find the redundancy vector $F=(f_0,f_1,....,f_{L-1})$ which maximizes received quality / minimizes distortion based on a given packet loss rate.

$$0 \le f_i \le f_{i-1} \le K-1$$

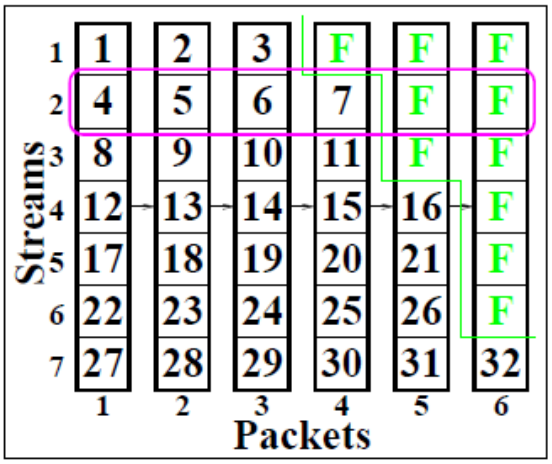

Figure 6: Streams divided across packets (From [Mohr1999])

Given a quality index $g()$ we seek

$$F_\theta = \arg\max_F \sum_{l=0}^{L-1} c(f_l,\theta) g_l(F)$$

Where θ is packet loss rate, $c(f_l, \theta)$ is probability of recovery based on stream l's error coding level, and $g_l(F)$ is the incremental quality achieved by decoding the message bytes $M_x..M_y$ assigned to stream l under strategy F.

As discussed in Chapter 2, The EFIC compression algorithm is a modification of SPIHT with a wavelet coefficient weighting to prioritize coefficients in order of perceptual importance. In traditional SPIHT, wavelet coefficients are encoded in order of bit planes. For each bit plane b, we consider the set of active coefficients, A_b, which contains all coefficients with their largest magnitude bit occurring on b union with the set of active coefficients from greater magnitude bit planes. We consider a modification of

Said & Pearlman's distortion measure [Said1996] to consider each bit of each transmitted coefficient:

$$D_{mse}(p-\hat{p}) = \frac{1}{N}\sum_{b=15}^{0}\sum_{a\in A_b}(2^b a(b) - 2^b \hat{a}(b))^2$$

Where a is a wavelet coefficient from the current active set, $a(b)$ is the b^{th} bit of the coefficient in the original image and $\hat{a}(b)$ is the b^{th} bit of the recovered/transmitted coefficient.

EFIC transmits perceptually weighted coefficients, and we can construct a similar distortion measure as distortion relative to a foveation weighted quality index (FWQI):

$$D_{FWQI}(p-\hat{p}) = \frac{1}{N}\sum_{b=15}^{0}\sum_{a\in A_b}(2^b f(b) - 2^b \hat{f}(b))^2$$

Where we consider scaled coefficient bits, $f(b)$ and $f'(b)$, rather than direct coefficient bits. Using this FWQI we can see that all coefficient bits on the same bit plane have the same incremental distortion value, which is 4 times greater than the bits from the next lower plane.

EFIC/SPIHT progressively encode the image in order of bit planes, creating the image message M. We can divide M into B segments, $M = \{M^{B-1}, M^{B-2}, \ldots, M^0\}$, where $M^b = \{M_x, M_{x+1}, \ldots\}$ is the set of bytes encoding the b^{th} bit plane. We can then assign a perceptual weight to each member of M^b according to:

$$g(m \in M^b) = g(b) = \frac{4^b N(A_b)}{N(M^b)}$$

Where $N(A_b)$ is the number of elements in the active set at the particular bit plane, and $N(M^b)$ is the total number of bytes required to encode the bit plane.

To model *g(b)*, we analyzed four 300 frame video sequences (see Chapter 5) with a resolution of 352 x 288 pixels and calculated the average perceptual importance of each byte across individual sequences and for the entire dataset. The sequences used are: clips of a news anchor (S0), a coast guard vessel (S1), a speaking foreman (S2), and a container ship (S3). Our results are shown in Figure 7.

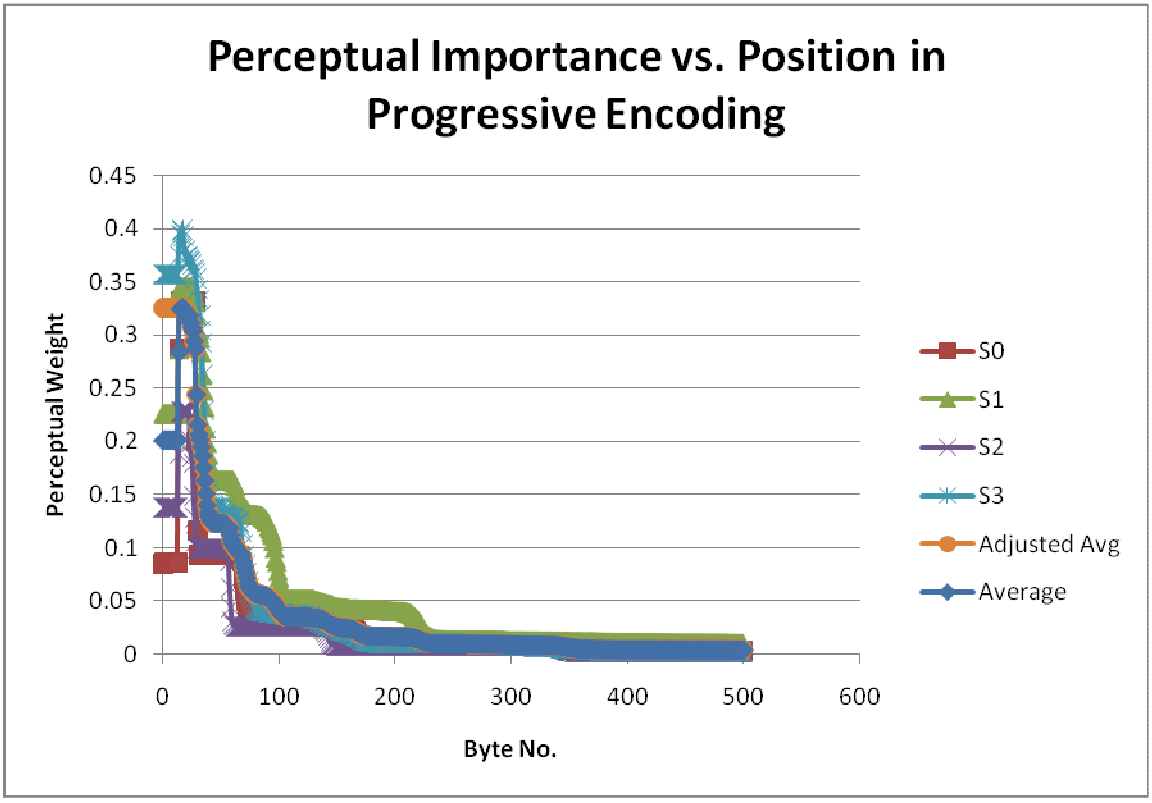

Figure 7: RFCode perceptual importance vs. position in byte stream

Across all four sequences, the perceptual importance values follow very similar trends. A common model should be able to adequately address all similar scenes transmitted using RFCode. Scenes will require use of a different model if their resolution is significantly different from 352 x 288, and if the foveation parameters used are adjusted significantly.

One interesting aspect of this analysis is that our method assigns a lower importance to bytes encoding the highest magnitude bit plane (bytes 0 to approximately 15) than the second highest magnitude bit plane. The highest magnitude bit plane is entirely composed of new coefficients and initiates the spatial tree, resulting in low coefficient per byte efficiency. Because decoding an image byte relies on all previously decoded bytes, a byte should not be assigned a lower perceptual importance than any subsequent bytes (we desire our modeled *g(m)* to be monotonically decreasing). We adjusted our model to meet this requirement by assigning each byte position the maximum importance of all subsequent bytes inclusive. The new adjusted average is shown in Figure 7 and is the model we used during optimization.

In support of the optimization algorithm, we evaluate the expected perceptual importance of a redundancy vector (*G(F)*) to be the sum of each byte's perceptual importance times the probability the byte will be recovered. The probability of recovery is based on a binomial distribution calculating the likelihood that dropped packets do not exceed the number of parity bytes in the stream.

$$G(F) = \sum_{l=0}^{L-1} \sum_{m \in M(l,F)} g(m) \sum_{i=0}^{f_l} \binom{n}{i} d^i (1-d)^{n-i}$$

Where *M(l,F)* is the set of message bytes assigned to stream l using strategy F, f_l is the number of parity bytes assigned to stream l, *g(m)* is the model of perceptual importance per byte position (see Figure 7), d is the probability of a dropped packet, and n is the total packet count (number of bytes per stream).

The algorithm we used is:

1. Initialize $F_{best} = (0,0,0,\ldots 0)$
2. Calculate $G_{best} = G(F_{best})$
3. Until no change in F_{best},
    a. $F_{search} = F_{best}$

b. For each stream l, and search range adj = -2 .. 2

i. $F_{cur}(l) = F_{search}(l) + adj$

ii. $G_{cur} = G(F_{cur})$

iii. If $G_{cur} > G_{best}$, $F_{best} = F_{cur}$

Using the above optimization algorithm, we derived ULP redundancy vectors for 352 x 288 video transmitted in a variety of conditions. The results for transmission over six 1200 byte packets are shown below in Figure 8.

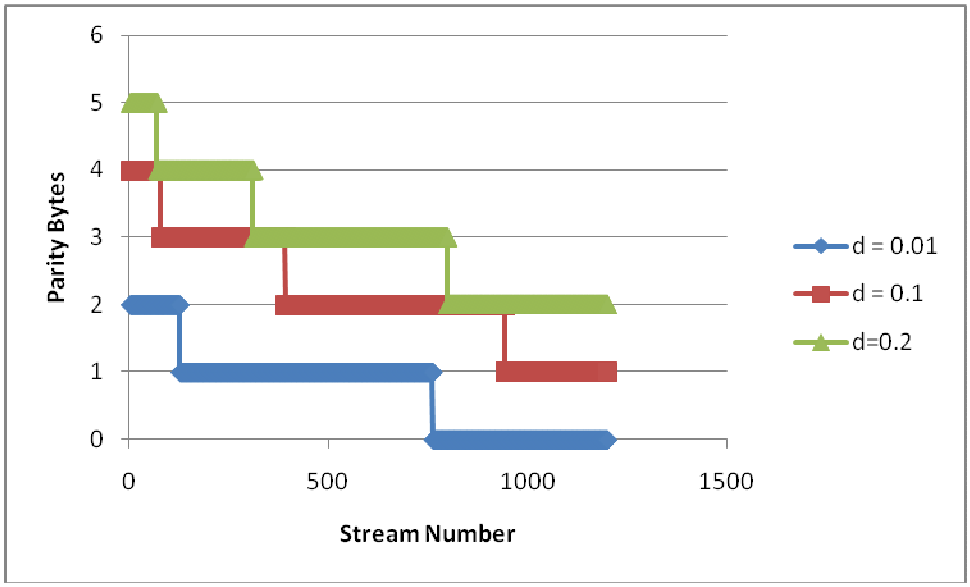

Figure 8: Optimized Redundancy Vectors for Six Packets

The above ULP optimization approach maximizes expected perceptual quality for each individual frame. The resulting video transmission has a vanishingly low probability of dropping even a single frame, although frame quality varies due to truncation at higher stream numbers. This is appropriate for applications such as remote operation or automatic tracking, with high sensitivity to temporal video quality – where motion jitter caused by a dropped frame is unacceptable. However, in applications without the same temporal restrictions, such as video conferencing, the frame to frame

quality changes caused by partial recovery may be more perceptually unsatisfying than a jerky transmission.

We developed an alternative form of the G(F) perceptual quality objective function to address the varying importance of temporal fidelity versus repeating a higher quality previous frame.

$$G(F) = \sum_{l=0}^{L-1} \sum_{m \in M(l,F)} g(m) \sum_{i=0}^{f_l} \binom{K}{i} d^i (1-d)^{K-i} + c \sum_{l=0}^{L-1} \sum_{m \in M(l,F)} g(m)$$

The second term biases the optimization towards maximizing the number of transmitted bytes (optimizing with this term alone results in the ELP strategy discussed at the beginning of this chapter). The value c is a constant used to select the importance of temporal fidelity. We optimized parity strategies for three different frame drop costs: high ($c = 0$, the original ULP cost function), moderate ($c = 0.5$) and low ($c = 1.0$). Figure 9 compares the different schemes for transmission using six packets at a packet loss rate of 0.2.

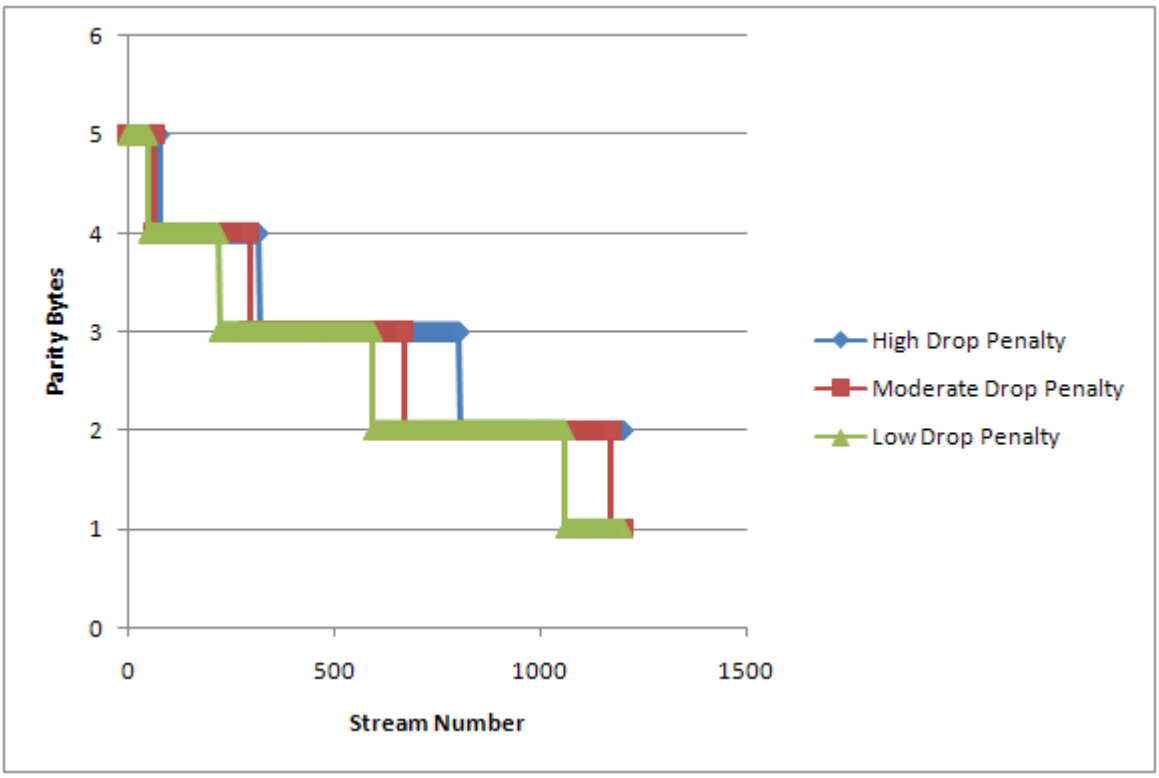

Figure 9: ULP Parity Schemes at different frame drop penalties

One additional detail of the ULP strategy is that the above derivation describes transmission of a gray-scale image. RFCode, however, supports transmission of color video in the YUV color space. Each color channel is assigned a portion of the byte budget, 80% for the intensity (Y) channel, and 10% each for the two chrominance channels (U & V). To address color video we assign each color channel an independent set of streams and apply the same ULP strategy to each.

Chapter 5: Prototype Codec Evaluation

5.1 RFCode Prototype

We implemented a prototype version of the RFCode within a componentized framework for simulation and evaluation of video codecs. Using this test framework, we evaluated the RFCode prototype and a baseline MJPEG algorithm using identical interfaces and data.

Our simulation framework models a packet based video transmission scheme over a lossy network, as shown in Figure 10. The framework components are:

- **Video Source.** Loads uncompressed image frames.
- **Compression Algorithm.** Converts image to compressed representation.
- **Packetization.** Divides compressed image into packets.
- **Simulated Channel.** Transmits packets, dropping some based on configured loss rate.
- **Packet Recovery.** Attempts to restore compressed image representation.
- **Decompression Algorithm.** Reverses compression process.
- **Video Display & Metrics.** Displays full resolution image and records video quality and bandwidth metrics.

The RFCode and MJPEG baseline approaches provide different implementations of the compression, packetization, decompression, and packet recovery algorithms.

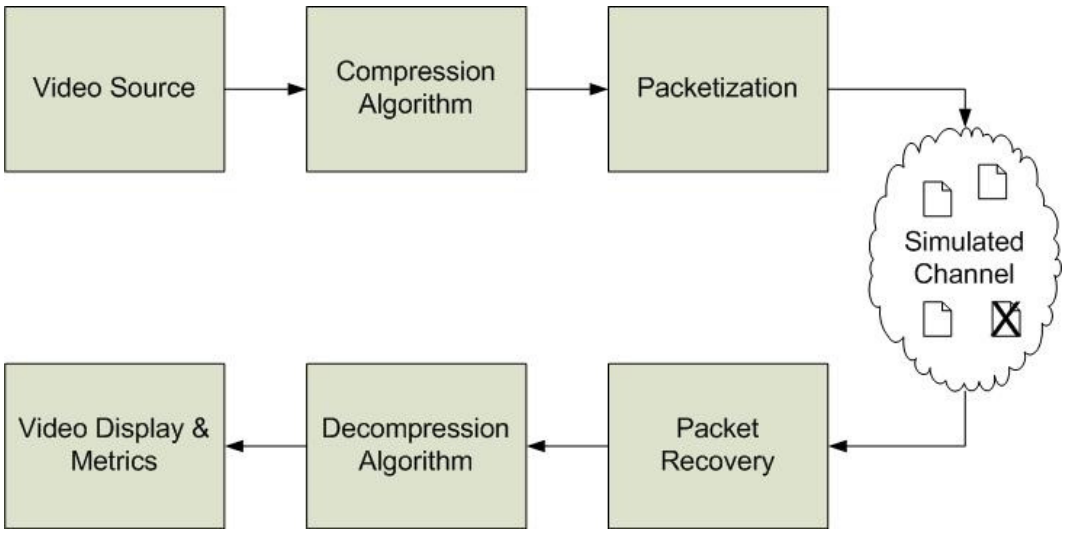

Figure 10: RFCode Codec Simulation Framework

We perform channel simulation using files on the hard-drive. The packetization approach divides the result of the compression algorithm into a number of files, each representing a single packet and with a size no greater than the configured MTU. The channel then selectively identifies which 'packets' to successfully transmit using a uniformly distributed random variable to achieve the desired packet loss rate. The packet recovery algorithm must then reconstruct the compressed image from the available packets, or flag that a frame has been dropped.

The RFCode prototype algorithm combines the EFIC compression algorithm (Chapter 2) with a packetization strategy using a Reed-Solomon error correcting code (Chapter 4).

We based our EFIC algorithm around C source code from the authors of [Wang2001]. We made minor modifications to the provided code to support variable image size, adjustable fovea placement, and to accept a wider variety of input formats.

We integrated the EFIC code into our experiment framework by constructing a Dynamic-Link Library (DLL) compatible with Sun's Java Native Interface (JNI) standard.

The principle classes of our Java-side implementation of EFIC are shown in Figure 11. The EFICConverter class acts as glue code to support our color EFIC extension and to provide the ICompressor interface required by our experimental setup. The EFICConverter allows direct selection of a desired encoding rate in bits per pixel. It also allows selection of the color channel bandwidth ratio between Y, U, and V channels in the encoded image.

PeerEFICImageBW provides access to a C-side image class with wavelet decomposition support. The PeerEFICDec and PeerEFICEnc classes link to a C-side encoder and decoder capable of compressing an image to disk and recovering from disk.

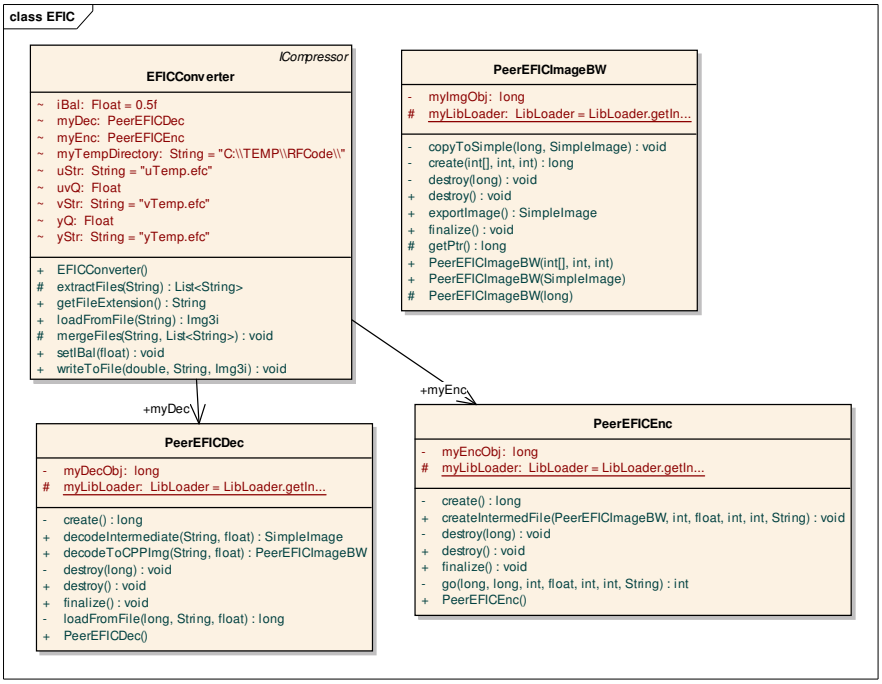

Figure 11: UML diagram of RFCode's EFIC interface

Similarly, the RFCode Reed-Solomon algorithm is a slight modification of existing third-party source code. We incorporated a RS implementation available under

the GNU Lesser General Public License (LGPL) at http://www.ka9q.net/code/fec/ into our prototype system through the JNI.

The principle classes of our Java-side implementation of Reed Solomon encoding are shown in Figure 12. The Packetizer class supports multiple operation modes, including RS(255,223) based variable parity protection as described in Chapter 4. This operation converts an input file into a number of files no larger than the configured MTU, representing packets as required by our video codec simulator. The PeerRSCode provides encoding and decoding support for byte arrays through the JNI.

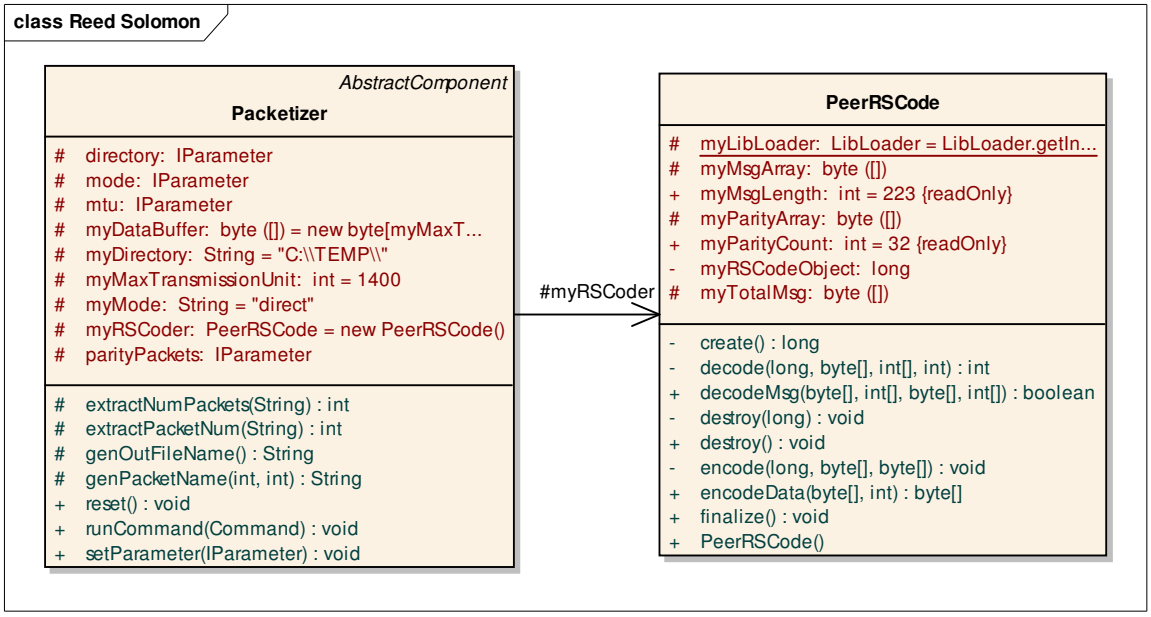

Figure 12: UML diagram of RFCode's RS interface

5.2 MJPEG Baseline

Our performance analysis compares RFCode performance with that of a baseline motion JPEG (MJPEG) codec. To support this analysis, we developed a simple MJPEG codec compatible with our video codec simulator. The MJPEG codec transmits individual

video frames compressed as JPEG images. MJPEG is more resilient to packet loss than intra-frame codecs such as MPEG-4, but provides poorer image quality at high levels of compression. The RFCode codec was designed to provide both superior image quality and superior resilience to packet loss than MJPEG.

We supported the codec simulator's ICompressor interface with a wrapper class using Sun's JPEG codec (bundled with the Sun's Java Development Kit). Compression rate is controlled by setting the codec's quality field. Although not linearly related to bit-rate, this field allows us to increase or reduce the relative compression of the encoder.

Our MJPEG baseline utilizes a simple packetization strategy, dividing the compressed image into the minimum number of packets without additional error correcting codes. This allows for maximum bandwidth devoted to image quality, but can increase the rate of dropped frames, as all packets must be successfully transmitted. When a frame is lost due to packet drops, the MJPEG baseline repeats the last received frame (zero-order hold). Our RFCode approach also uses zero-order hold for missing frames, but drops fewer frames due to its error correcting code.

5.3 Evaluation Data

For our evaluations we analyzed codec performance on video sequences with resolution and video characteristics similar to the expected RFCode operating environment. These sequences are part of a common corpus of test data for video processing algorithms, and are available via the internet from the site http://media.xip.org/video/derf/.

All sequences used have a full-resolution of 344 x 288 pixels, a color bit-depth of 24 bits per pixel, and a duration of 300 frames. The sequences are:

- **Coastguard.** The coastguard sequence pans to follow the progress of a coastguard ship in a littoral environment. The subject matter of interest remains centered throughout the sequence and there are significant changes in background.

Figure 13: Coastguard sequence representative frame

- **Container.** The container sequence is filmed from a stationary viewpoint and shows a cargo ship and escort moving out to sea.

Figure 14: Container sequence representative frame

- **Foreman.** The foreman sequence is shot with a handheld camera and shows a man wearing a hardhat giving an animated speech. The subject matter is well centered, and the sequence shows strong dynamic content.

Figure 15: Foreman sequence representative frame

- **Newscaster.** Also known as the 'Akiyo' sequence, the newscaster is shot with a stationary camera and shows a woman reporting the news.

Figure 16: Newscaster sequence representative frame

- **Football.** The football sequence follows a collegiate football play. The area of greatest interest remains generally centered throughout the sequence. This is a very dynamic sequence with action occurring throughout all portions of the image.

Figure 17: Football sequence representative frame

- **Harbor.** The harbor sequence is shot from a stationary view point. It shows several ships traversing a marina channel from right to left.

Figure 18: Harbor sequence representative frame

- **Husky.** The husky sequence pans to follow a runner leading a dog. The sequence is very dynamic and the subject area fills much of the frame.

Figure 19: Husky sequence representative frame

- **Ice-skating.** The ice-skating sequence is shot from a stationary perspective. It shows many people ice-skating and contains much action throughout the field of view.

Figure 20: Ice-skating sequence representative frame

5.4 EVALUATION METRICS

We evaluated each codec's performance using a set of metrics that capture its level of compression and visual distortion. For each experimental analysis we recorded the metrics:

- **Bits per pixel (bpp).** This number represents the total data requirements of a transmitted frame, including any parity (error correction) information. We divide the total number of transmitted bits by the number of pixels. Referring back to Table 1, we are particularly interested in performance rates in the region of 0.4 bpp.
- **Packet drop rate.** The probability that the transmission channel will drop a packet. All packets have an equal chance of being dropped. This is a configurable value selected for each experiment.
- **Frame drop rate.** The number of frames dropped divided by all frames transmitted. Equivalently, the percentage of frames dropped.
- **Mean Squared Error (MSE).** We apply this traditional quality metric across all color channels in an RGB representation, for all pixels and all frames. Low MSE indicates better performance.

$$MSE = \frac{\sum_{n=1}^{N}\sum_{r=1}^{R}\sum_{c=1}^{C}(R'-R)^2 + (G'-G)^2 + (B'-B)^2}{N \cdot R \cdot C}$$

- **Mean Squared Error (MSE), no drops.** This metric records MSE values in a codec simulation with zero probability of packet loss. This measures the still-image quality of the codec without considering the impact of motion artifacts due to frame drops.
- **Average Foveated Structural SIMilarity (F-SSIM).** We apply the industry-leading SSIM metric with a foveated weighting scheme to the intensity channel of the transmitted images. SSIM estimates structural similarity for quality assessment; it is a

combination of terms measuring deviation in mean luminance, rough contrast, and structure [WangBovik2004]. The Foveated SSIM (F-SSIM) reweights SSIM values based on their perceptible contribution to error, similar to the weighting described in Section 3.2.2 [HaBovik2009]. We average F-SSIM values for all frames. F-SSIM ranges from 0 to 1, with higher values indicating greater similarity. In presentation, we convert to an error metric by plotting (1 - F-SSIM). Like MSE, lower values of (1 - F-SSIM) indicate better performance.

- **Foveated Structural SIMilarity (F-SSIM), no drops.** As above, but calculated only for successfully transmitted frames. This removes error caused by dropped frames.

These metrics are best interpreted as performance curves displaying performance relative to an external parameter such as bandwidth (equivalently bits per pixel). We use the following curves to compare codec performance:

- **Rate-distortion curves.** These curves show the overall error (measured by either MSE or F-SSIM) at different compression levels. The curves plot MSE or 1 – F-SSIM versus bits per pixel. Better algorithm performance is indicated by a rate-distortion curve below the competing approaches. One key consideration in this curve is the effect of frame drops. Increasing the bit rate can also increase the likelihood of dropping a frame. If our codecs are unable to reconstruct a transmitted frame due to packet loss they repeat the last frame (zero-order hold). This increases error by a variable amount depending on scene dynamics. In scenes with significant motion, error can increase as bit rate increases.
- **Rate-distortion ignoring frame drops.** These curves show expected performance over a perfect channel that drops zero packets. This allows direct comparison of the

compression algorithms without consideration for the effects of error correcting codes and other data recovery approaches.

- **Frame drop rate.** This curve plots frame drop rate versus bits per pixel, and demonstrates the expected frame drop rate for a given image bandwidth. As discussed in Section 3.2.3, we expect these curves to take the form of an inverse binomial distribution.
- **Error versus channel reliability.** These curves show error versus the channel's packet loss rate for a particular constant bandwidth. They illustrate the impact frame loss and video "stutter" have on video quality. These curves will help us select the best number of RFCode parity packets to use for a known channel bandwidth and loss rate.

5.5 EVALUATION PROCEDURE

We performed three sets of analysis using the video codec simulator framework described above. The first analysis directly compared the rate-distortion of the RFCode and MJPEG codecs over a lossless transmission channel. The second analysis compared the rate-distortion of the RFCode codec at varying parity settings with the MJPEG codec over a lossy channel with varying packet loss rates. The third analysis compared the performance of two versions of the RFCode codec with different parity strategies: one with equal loss protection (ELP), the other with unequal loss protection (ULP).

The video transmission without packet loss experiment compared the performance of:

- **MJPEG.** The baseline compression approach described in 5.2. No error correcting code was used.

- **RFCode.** The modified color EFIC algorithm described in 5.1. No error correcting code was used.

These codecs were applied to the 'Coastguard' sequence.

The video transmission with packet loss experiment compared the performance of:

- **MJPEG.** The baseline compression approach described in 5.2. No error correcting code was used.
- **RFCode, 1 Parity.** The modified color EFIC algorithm described in 5.1. The packetization strategy with Reed-Solomon error correcting code, as described in Chapter 4, was used with one parity packet.
- **RFCode, 2 Parity.** As above, but with RS configured for 2 parity packets.
- **RFCode, 3 Parity.** As above, but with RS configured for 3 parity packets.

These codecs were applied to the 'Coastguard,' 'Container,' 'Foreman,' and 'Newscaster' sequences.

During each analysis the competing codecs were exercised over the same range of bit-rates. Each codec was evaluated using an identical simulation framework for the same video data.

The loss protection strategy experiment compared the performance of:

- **RFCode, ELP.** The modified color EFIC algorithm described in 5.1. The packetization strategy with Reed-Solomon error correcting code, as described in Chapter 4, was used with parity selected to maximize the expected number of image data bytes recovered based on available number of packets and expected packet loss rate.

- **RFCode, ULP.** The modified color EFIC algorithm described in 5.1. The packetization strategy with Reed-Solomon error correcting code applied in a ULP strategy as described in Chapter 4.

These codecs were applied to the 'Coastguard,' 'Container,' 'Foreman,' 'Newscaster,' 'Football,' 'Harbor,' 'Husky,' and 'Ice-skating' sequences.

During each analysis the competing codecs were exercised over the same range of bit-rates and packet loss rates. Each codec was evaluated using an identical simulation framework for the same video data.

Chapter 6: Evaluation Results and Discussion

6.1 Performance Analysis without Packet Loss

Our first experiment compared the performance of RFCode's modified EFIC compression algorithm with the baseline MJPEG algorithm, disregarding packet loss. No RFCode bandwidth was allocated for parity packets in this experiment.

We analyzed the Coastguard sequence, described in 5.3, using varying levels of compression (bits per pixel) and measured the resulting distortion. The results of this experiment are presented below in Figure 21.

As desired, the RFCode codec significantly outperforms the MJPEG algorithm at very low bit rates. As indicated in Table 1, an appropriate compression algorithm must be able to achieve good performance at less than 0.4 bits per pixel. RFCode's superior performance at this low bit rate allows us to allocate some bandwidth to provide parity protection against packet loss while achieving superior image quality, as demonstrated in the second set of experiments.

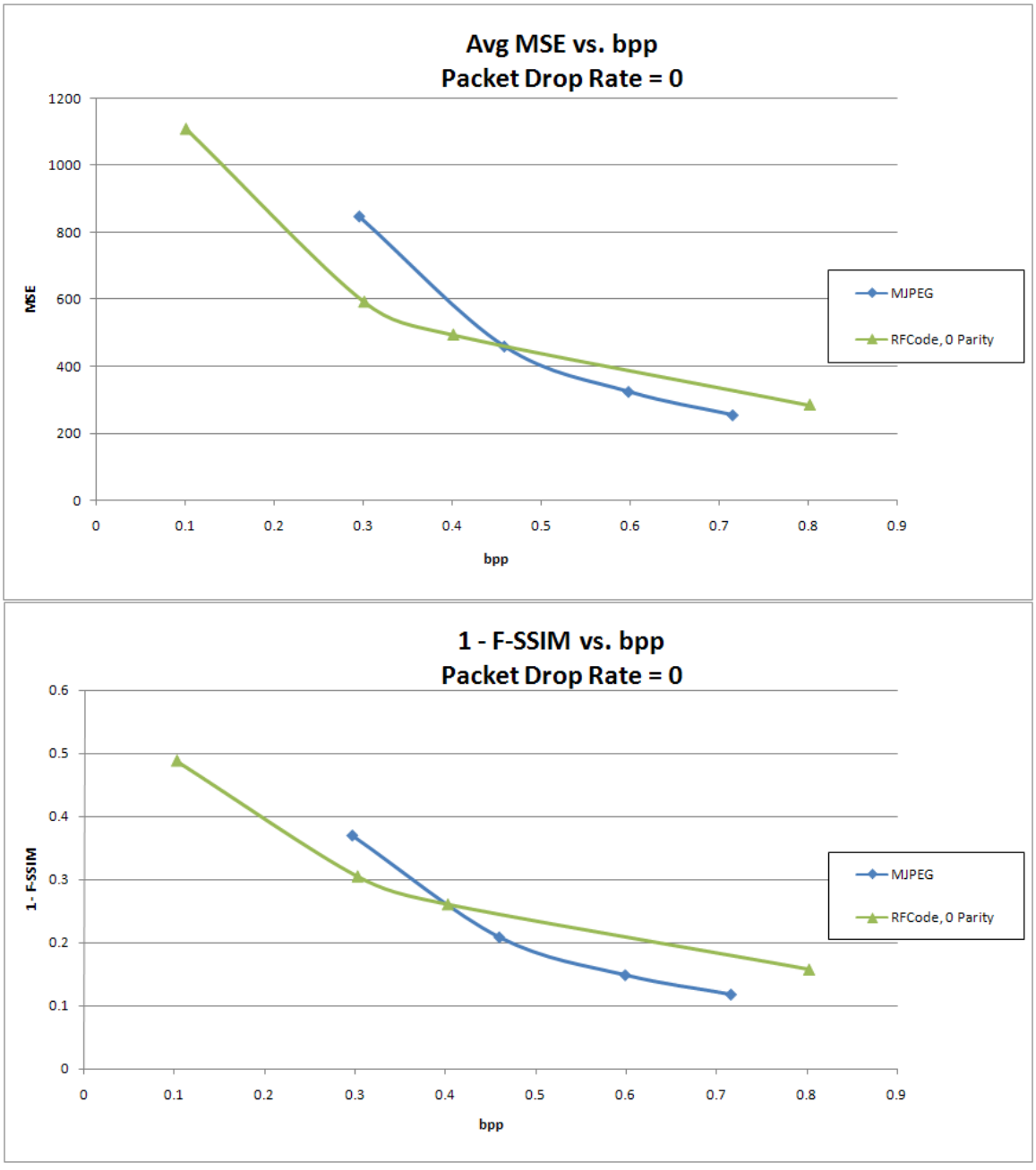

Figure 21: Rate-distortion curves, MSE (top) and F-SSIM (bottom) with no packet loss.

6.2 Performance Analysis with Packet Loss

The second experiment set we performed compared codec performance over a transmission channel subject to packet loss. For these experiments we enabled RFCode's RS error correcting code to provide 1, 2, or 3 parity packets. The results of these experiments are presented below.

We evaluated the first four test sequences described in 5.3 during these experiments. Each sequence was evaluated twice with randomized packet loss. We applied varying degrees of compression for each approach (MJPEG and RFCode with 1, 2, or 3 parity packets) and generated rate distortion curves from the measured error metrics.

Figure 22 shows the rate distortion curves from this experiment at the expected packet loss rate of 0.1. At this packet loss rate, the RFCode codec configured to transmit one parity packet per frame outperforms the MJPEG codec at all evaluated compression levels. The additional protection from packet loss provided by transmitting two or three parity packets turns out to be unnecessary. An interesting effect in this performance curve is that the MJPEG performance actually deteriorates as more bandwidth is used per frame. This is due to the increasing number of packets required to transmit frames at higher bandwidths. Because MJPEG has no means to recover from packet loss, the video stream jerks when this occurs, incurring a significant error penalty. RFCode, on the other hand, is able to make maximum use of the available bandwidth due to its inclusion of error correcting codes.

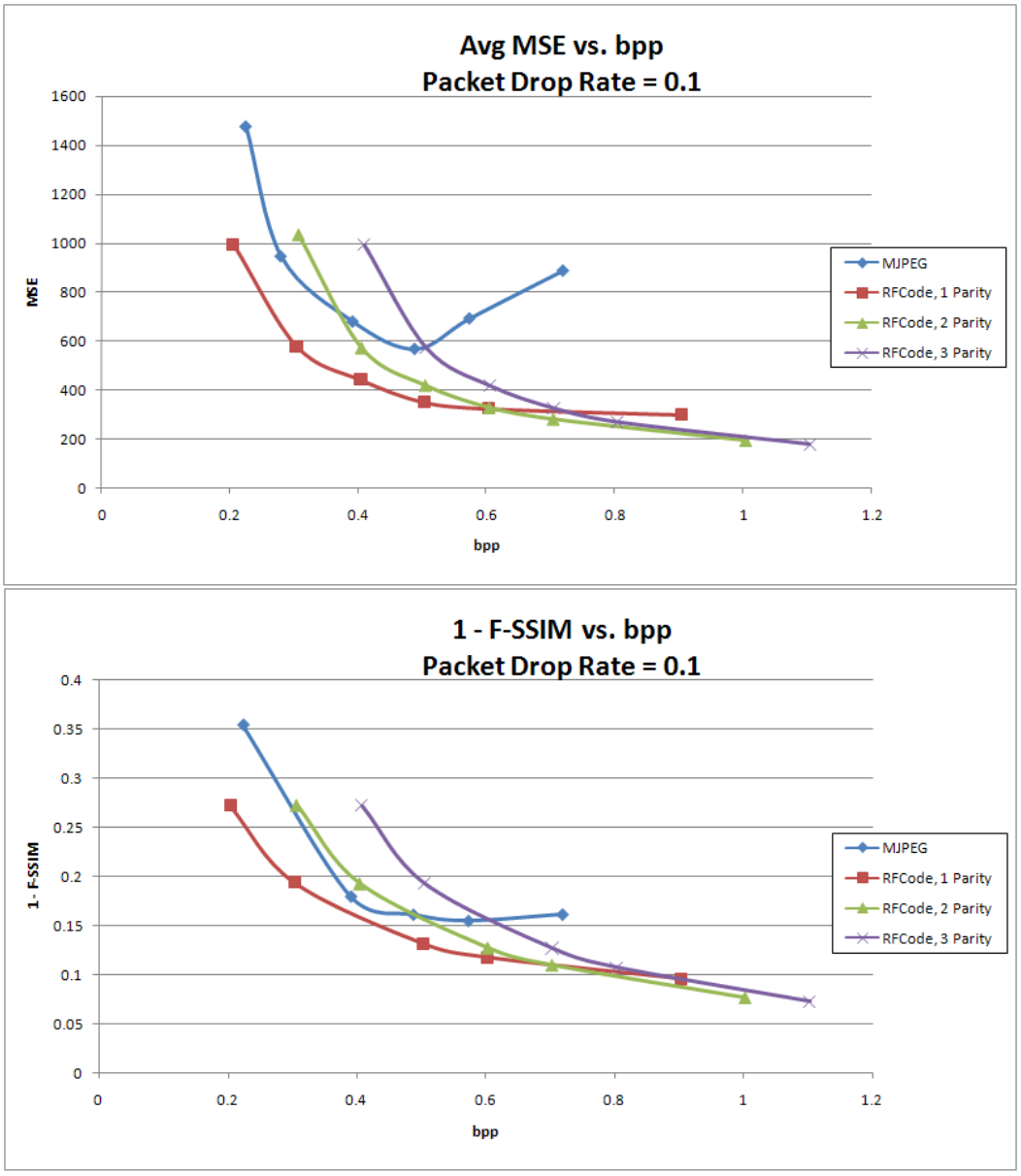

Figure 22: Rate-distortion curves using MSE (top) and F-SSIM (bottom) metrics for RFCode and MJPEG codecs with loss.

Figure 23 plots the relationship between frame drops and packet drops for each codec at a constant bit-rate. As explained in Section 3.2.3, this relationship is determined by a binomial distribution. By tolerating one, two, or three dropped packets per frame, the RFCode configurations provide significantly smoother video than MJPEG, even at twice the expected packet loss rate.

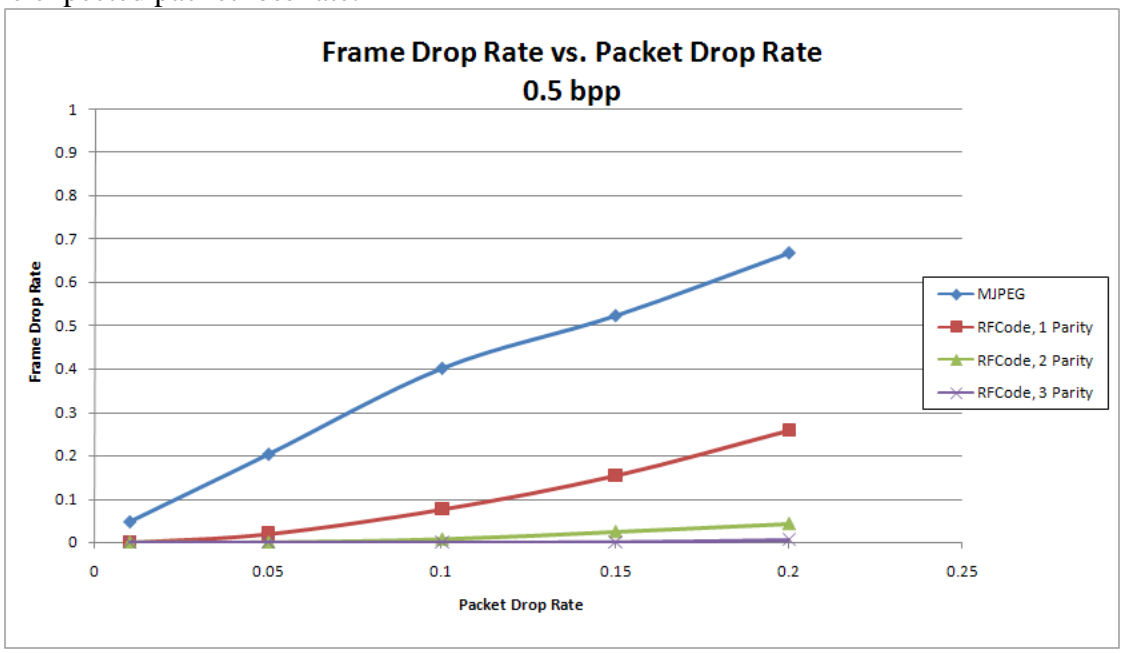

Figure 23: Frame drop rate vs. packet drop rate for RFCode and MJPEG codecs at constant bit rate.

Figure 24 translates the results of Figure 23 into an error metric, showing how dropped frames increase MSE. When a frame is not successfully transmitted, the video codec simulator repeats the last successfully transmitted frame. In scenes with significant movement this results in a large increase in MSE. The RFCode curves show an unexpected result as they indicate that MSE actually decreases when the packet loss rate increases from 0.15 to 0.2. As Figure 23 indicates, frame drops are very rare events

which have a significant impact on overall MSE. We believe that this unexpected performance may be due to the small sample size of dropped frames.

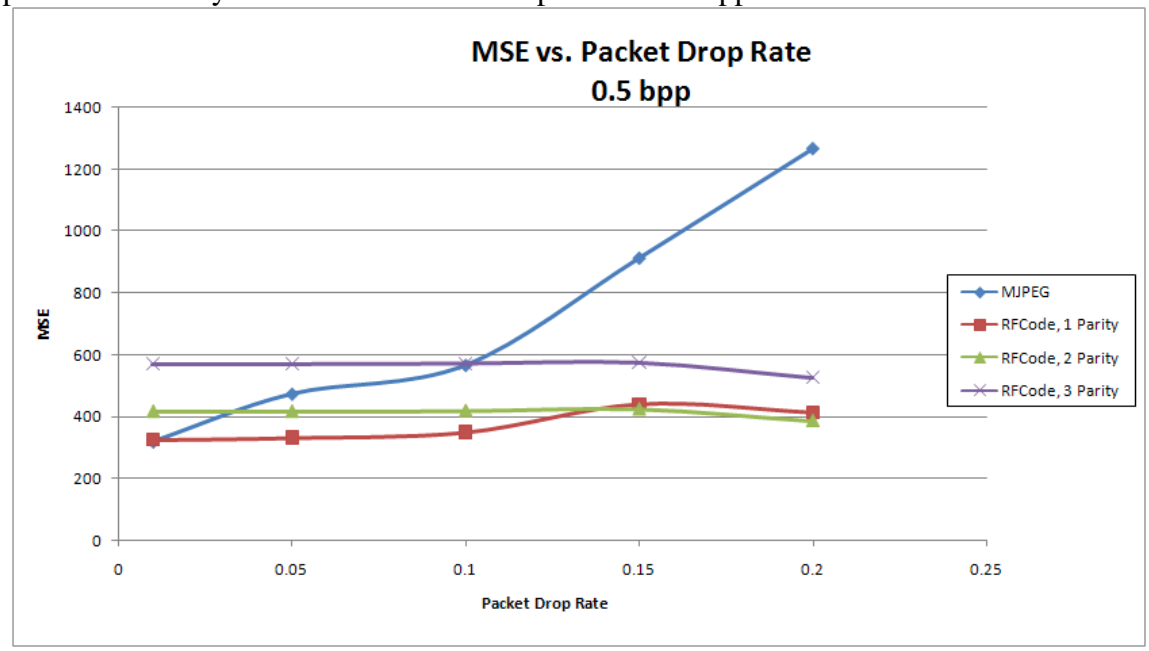

Figure 24: MSE vs. packet drop rates for RFCode and MJPEG codecs at constant bit rate.

6.3 Performance Analysis of ELP vs. ULP

The third experiment set we performed compared codec performance over a transmission channel subject to packet loss using either an Equal Loss Protection (ELP) strategy or an Unequal Loss Protection (ULP) strategy (see Chapter 4). Both codecs evaluated employed the general, foveated RFCode compression algorithm. The results of these experiments are presented below.

We evaluated all eight test sequences described in 5.3 during these experiments. Each sequence was evaluated twice with randomized packet loss. We applied varying degrees of compression for each approach (each frame compressed to four, five, or six 1200 byte packets) and generated rate distortion and frame drop rate curves from the measured error metrics.

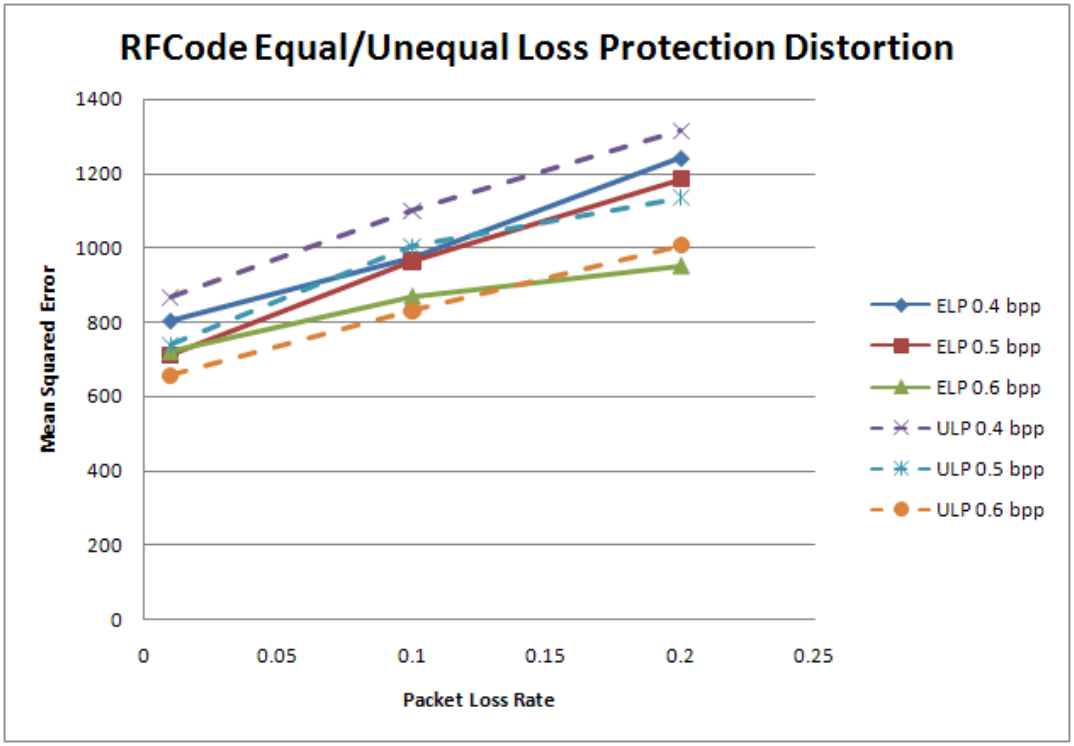

Figure 25: Average performance of ELP and ULP strategies

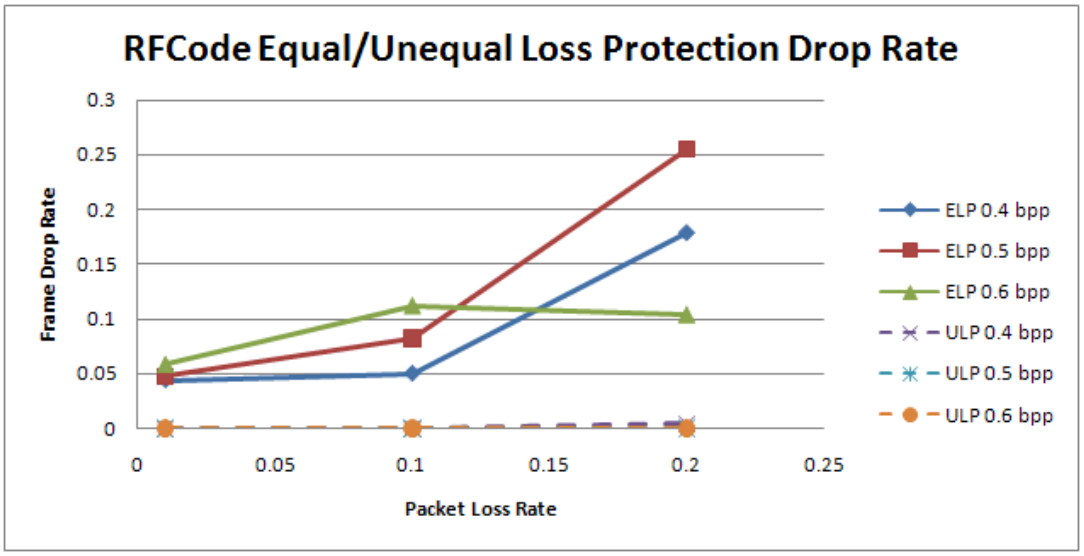

Figure 26: ULP and ELP frame drop rates

All ULP strategies achieve a significantly lower drop rate. This is very important for time critical applications where the 'motion jitter' caused by dropped frames results in unacceptable performance. On deeper analysis, we realized that ULP was optimized for these cases – the strategy objective function does not account for the potential value of repeating dropped frames rather than providing high levels of parity protection. In our derivation dropped frames are assigned zero perceptual value; however for slow moving sequences the MSE caused by a repeated frame is significantly less than that caused by a blank frame.

Based on this observation, we selected three action sequences from the eight test videos and three 'stationary' sequences. Analyzing results on these subsets, we see that ULP performs better on actions sequences (Figure 27), while ELP performs better on stationary sequences (Figure 28).

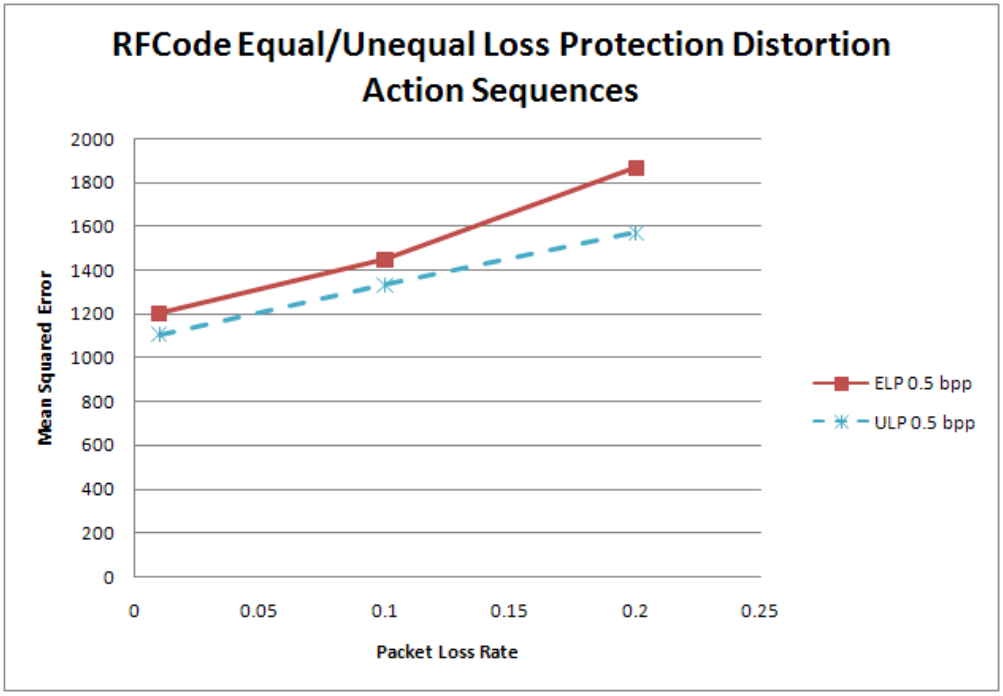

Figure 27: ULP and ELP performance for action sequences

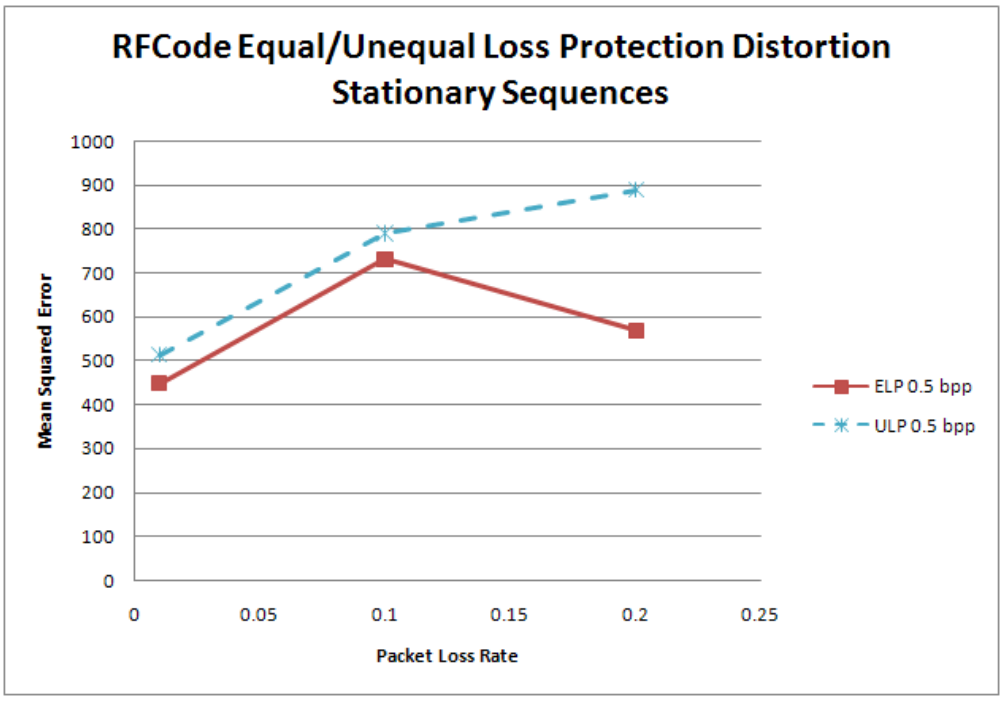

Figure 28: ULP and ELP performance for stationary sequences

Chapter 7: Conclusion

As our results indicate, the RFCode codec significantly outperforms the MJPEG baseline approach in a simulated low bandwidth video transmission application. This is due to two factors: the EFIC algorithm provides superior compression levels than JPEG compression, and the RFCode codec utilizes the resulting available bandwidth to provide robustness to packet loss through an error correcting code.

At the expected packet loss rate of 0.1, the RFCode codec with one parity packet has a frame loss rate nearly four times less than the MJPEG codec. This translates into remarkably smoother video. At the same time, this codec is able to provide significantly better video quality. Figure 29 compares two frames of the 'Coastguard' sequence from the MJPEG and RFCode codecs at the same bandwidth. Although the RFCode codec allocates 20% of the bandwidth to parity packets, it still provides a better image, particularly near the image center, which is assumed to be the area of greatest interest in these experiments.

Figure 29: Comparison of MJPEG (top) and RFCode (bottom) codecs at 0.5 bpp.

An interesting observation in our experiments was that the measured benefits of RFCode versus MJPEG were dependent on the content of the video sequence. Figure 30 shows the rate-distortion curves individually for the newscaster sequence and for the foreman sequence. When transmitting the foreman sequence, MJPEG achieves

comparable performance to RFCode with 1 parity packet. However, when transmitting the newscaster sequence, RFCode significantly outperforms MJPEG. The newscaster sequence is a near perfect fit for RFCode's foveated approach: nearly all high frequency information is contained near the image center, exactly where RFCode's Phase 1 fixation point is located. The foreman sequence, on the other hand, contains significant detail well outside the fovea location due to the subject occupying such a large part of the frame, and the clearly focused background.

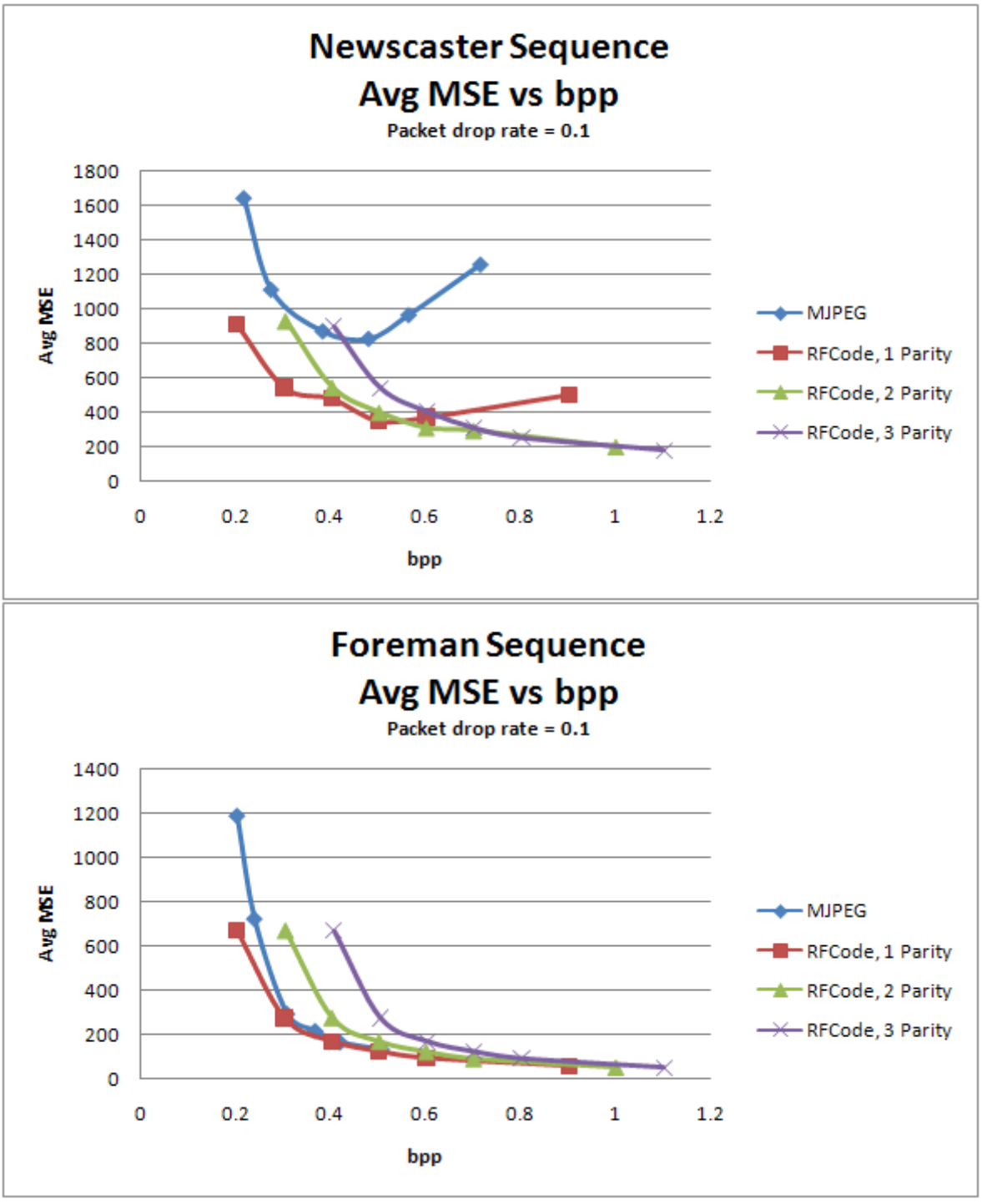

Figure 30: Rate-distortion curves for Newscaster (top) and Foreman (bottom) video sequences.

This result indicates that RFCode will have even greater benefits for applications like teleoperation, where the user is primarily interested in the center of the image field. In addition, by allowing dynamic configuration of foveation parameters, RFCode can be extended to provide superior results for an even broader range of scenarios.

We have successfully demonstrated that the RFCode approach can be used to achieve reliable video transmission over lossy packet-based tactical networks. We developed a novel video codec applying the techniques of foveated image compression and redundant encoding. A prototype implementation of this codec significantly outperformed a baseline MJPEG approach in quantitative experiments when transmitting video over simulated low bandwidth, unreliable networks.

A thorough series of experimental analyses demonstrated that the RFCode codec transmits significantly better video than a baseline MJPEG approach in a simulated environment modeled after a tactical network. In our experiments, RFCode was able to reduce frame drops due to unreliable network conditions by a factor of four, and improve Mean Squared Error (a measurement of video quality) by nearly a factor of two, when compared with an MJPEG codec using the same bandwidth.

The resulting RFCode video stream is operationally superior to the MJPEG video stream. At the same bandwidth, RFCode produces individually clearer images, particularly at the center of operator interest. The video stream as a whole also displays much smoother motion due to its resilience to packet loss. We have provided a framework for adjusting the codec's prioritization of spatial quality and temporal quality, allowing the approach to be employed in a variety of operational environments.

References

[Mohr1999] A. Mohr, E. Riskin, and R. Ladner, "Unequal Loss Protection: Graceful Degradation of Image Quality over Packet Erasure Channels through Forward Error Correction," *IEEE Journal on Selected Areas in Communication*, Vol: 18, 1999.

[Rajashekar2008] U. Rajashekar, I. van der Linde, A.C. Bovik and L.K. Cormack, "GAFFE: A gaze-attentive fixation finding engine," *IEEE Transactions on Image Processing*, Vol: 17 No: 4, April 2008.

[Reed1960] I. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *SIAM Journal of Applied Math*, 1960.

[Said1996] A. Said and W. A. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, June 1996.

[Shapiro1993] J. M. Shapiro, "Embedded image coding using zerotrees of wavelets coefficients," *IEEE Trans. Signal Processing*, vol. 41, pp. 3445–3462, Dec. 1993.

[Wang2001] Z. Wang and A. C. Bovik, "Embedded foveation image coding," *IEEE Trans. Image Process.*, vol.10, no.10, pp.1397-1410, Oct. 2001.

[Watson1997] A. B. Watson, G. Y. Yang, J. A. Solomon, and J. Villasenor, "Visibility of wavelet quantization noise," *IEEE Trans. Image Processing*, vol. 6, pp. 1164–1175, Aug. 1997.